applied sciences

MDPI

Article

A Robust Sharding-Enabled Blockchain with Efficient Hashgraph Mechanism for MANETs

Ruilin Lai *, Gansen Zhao, Yale He and Zhihao Hou

School of Computer Science, South China Normal University, Guangzhou 510631, China; gzhao@m.scnu.edu.cn (G.Z.) heyale@m.scnu.edu.cn (Y.H.); houzhihao@m.scnu.edu.cn (Z.H.)
* Correspondence: 2020010184@m.scnu.edu.cn

Abstract: Blockchain establishes security and trust in mobile ad hoc networks (MANETs). Due to the decentralized and opportunistic communication characteristics of MANETs, hashgraph consensus is more applicable to the MANET-based blockchain. Sharding scales the consensus further through disjoint nodes in multiple shards simultaneously updating ledgers. However, the dynamic addition and deletion of nodes in a shard pose challenges regarding robustness and efficiency. Particularly, the shard is vulnerable to Sybil attacks and targeted attacks, and dishonest gossip reduces the efficiency of hashgraph consensus. Therefore, we proposed a behavior-based sharding hashgraph scheme. First, dishonest behaviors of nodes are recorded in a decentralized blacklist. Gossip information is sent to a reliable neighbor, and gossip information from another reliable neighbor is received. Second, a tree-assisted inter-sharding consensus is proposed to prevent Sybil attacks. The combination of shard recovery and reconfiguration based on node state is devised to prevent targeted attacks. Finally, we conducted the performance evaluation including security analysis and experimental evaluation to reveal the security and efficiency of the proposed scheme.

Keywords: blockchain; MANET; hashgraph; sharding; reputation

Citation: Lai, R.; Zhao, G.; He, Y.; Hou, Z. A Robust Sharding-Enabled Blockchain with Efficient Hashgraph Mechanism for MANETs. *Appl. Sci.* **2023**, *13*, 8726. https://doi.org/10.3390/app13158726

Academic Editor: Gianluca Lax

Received: 21 June 2023
Revised: 23 July 2023
Accepted: 23 July 2023
Published: 28 July 2023

1. Introduction

Blockchain has attracted widespread attention from all over the world [1] because of its capability of establishing trust in a decentralized manner. Distributed ledger technology makes the recorded transactions immutable, transparent and traceable, such that the blockchain is applied to some IoT (Internet of Things) scenarios, e.g., VANET (vehicular ad hoc network) [2], UAS (unmanned aerial system) [3] and MCS (Mobile CrowdSensing) [4]. A mobile ad hoc network (MANET) is established autonomously by mobile devices without relying on any infrastructure, and the mobile devices communicate in wireless multi-hop mode [5]. The MANET is decentralized in communications. The integration of blockchain and MANET improves both MANET security and blockchain adaptability. Nowadays, the blockchain is explored to build on MANETs [6–9], and its applications involve incentive data collection [10], secure resources sharing [11–13], and an expansion of cryptocurrency [14,15]. A MANET-based blockchain is geographically closer to end users than the blockchain based on edge computing, so it is envisioned to provide more safety and privacy protection.

Hashgraph [16] is more applicable to the MANET-based blockchain compared with current popular consensus mechanisms. Since Proof-of-X, PBFT (Practical Byzantine Fault Tolerance) [17] and RAFT [18] require one miner/leader for transactions collection and block generation, they conflict with the decentralized characteristics of the MANET in communications, reducing the security and efficiency of the blockchain. In other words, it is hard to select a reliable miner/leader which has the minimum transmission delay to all other nodes in a dynamic network. The PoW (Proof of Work) [19] aggravates the computational burdens of the nodes with limited resources. Under opportunistic communications incurred by node mobility, the collection of vote messages in the PBFT or RAFT is

in low rates, resulting in latency of the blockchain update. Even worse, the communication complexity $O(n^2)$ of the PBFT takes up a lot of network resources. Fortunately, hashgraph has the capacity of supporting the dynamic nature of MANETs. It is because: (i) neighbor nodes in the MANET are dynamically changed, while the leaderless and asynchronous operations of hashgraph avoid the overhead of route discovery and construction, e.g., the route from a miner/leader to other nodes. So, hashgraph mitigates the latency caused by neighborhood changes. (ii) When network partition occasionally occurs in the MANET, an isolated group of nodes can continue to perform virtual voting in hashgraph. Once the network merges, votes are counted to confirm events. (iii) The wireless communication resources of the MANET are limited, while gossip about gossip and the virtual voting mechanisms of hashgraph make each gossip contain votes for multiple received gossips, without extra votes transmission, improving the transmission efficiency of voting messages. Intuitively, by abstracting the physical network topology [20], the communication links of the above consensus protocols form the logical topologies of blockchain networks in Figure 1. Hashgraph, which is the mesh network without a leader, is similar to the MANET.

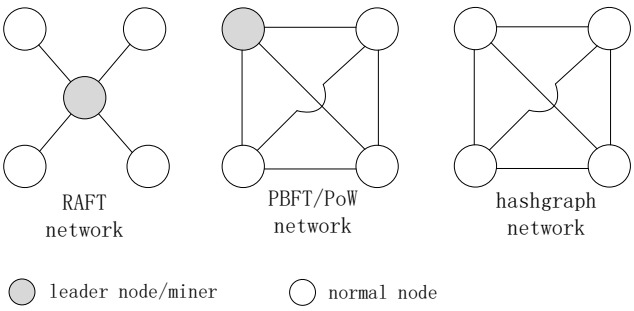


Figure 1. Logical topologies of blockchain networks with different consensus protocols.

Sharding technology scales the blockchain effectively through the parallel execution of multiple shards. The blockchain nodes and the distributed ledger are divided into shards. Two critical components of the sharding protocol are defined as intra-sharding and inter-sharding consensus [21,22]. Within a shard, nodes are responsible for updating transactions to the local ledger via the intra-sharding consensus. The inter-sharding consensus guarantees the atomicity and consistency of cross-shard transactions. The atomicity means the cross-shard transaction is committed and aborted atomically, and the consistency is that the transaction is either updated to the ledgers of all involved shards or not updated. Furthermore, if each shard is constructed by nodes with geographical proximity, it not only improves communication efficiency in the intra-sharding consensus but also reduces the number of cross-shard transactions related to geographic information.

Nevertheless, the dynamic addition and deletion of nodes make shards more vulnerable to being compromised, which is the motivation of our work. Reputation-based asynchronous consensus in VANETs alleviates these attacks to some extent [23,24]. Since we only consider the movement of nodes, the VANET is regarded as one kind of MANET. In Zhang et al. [23], nodes whose reputation values are lower than a given threshold are regarded as malicious ones, and the consensus threshold is calculated based on the number of malicious nodes within a shard. After a block is generated through PoW resolution, nodes add their signatures into the block for validation and propagate it to neighbor nodes. Only when the number of attached validation signatures of the block is larger than the consensus threshold is the block sent to a nearby RSU (Road Side Unit) for synchronizing to other nodes. Thus, even though malicious nodes come into a shard, the consensus threshold increases and the attack cost rises accordingly. In Wang et al. [24], nodes with high reputation values are randomly selected into a hashgraph committee, becoming a primary and primary candidates. In the process of hashgraph consensus, the primary packs validated events into a block. Then, the block is confirmed by accounting nodes based on

the majority, and it is recorded in the chain. The members of the hashgraph committee are fixed, while newcomers become accounting nodes dynamically. Since the primary and primary candidates are authenticated by a certificate authority and their voting weights are related to reputation scores, the Sybil attack can be prevented.

Therefore, we propose a behavior-based sharding hashgraph for nodes dynamically joining and leaving a shard in MANETs. However, it needs to overcome the challenges of robust shards and efficient hashgraph consensus. **Robustness**. The reputation of nodes should not be centrally managed by a third party, e.g., a trust authority [23] or a certificate authority [24]. It is because the single point is potential risks. Moreover, if a shard is compromised, a recovery mechanism is necessary to restore the transactions generated by honest nodes in the blockchain. **Efficiency**. The transmission delay is relatively high in MANETs due to dynamic changes of neighbor nodes [25]. Dishonest gossip, which is used to propagate false gossip or not to propagate received gossip to neighbors, aggravates the latency of blockchain updates.

In order to address the above challenges, our contribution is as follows:

- A sharding hashgraph framework with a blacklist is proposed for the MANET-based blockchain. In each shard, the blacklist records the dishonest consensus activities of nodes in a decentralized manner. With the assistance of the blacklist, gossip information is sent to a reliable neighbor, and gossip information from another reliable neighbor is received first. It improves the efficiency of gossip about gossip in hashgraph consensus.
- A tree-assisted inter-sharding consensus is proposed to prevent Sybil attacks. A tree, $tree^{je}$, whose root is the participation request of a node, is constructed by all involved shards. Based on $tree^{je}$, the requester with enough reputation values is allowed to join the shard, and its reputation values cannot be reused in other shards. Furthermore, transactions of the requester can be updated to the graph, and the requester propagates gossip information first. Once the requester is allowed to join the shard, its updated transactions can be used directly, and previous virtual voting is counted in hashgraph consensus. Hence, the requester can participate in activities of the shard without waiting.
- The combination of shard recovery and reconfiguration based on node state is proposed to prevent targeted attacks. Based on node state changes and $tree^{je}$, a compromised shard can be determined by other shards and then is recovered. When node states are changed frequently in the shard, the shard is reconfigured dynamically and adaptively.

2. Related Works

2.1. Sharding-Enabled Blockchain

The sharding technology enhances scalability of the blockchain through simultaneously maintaining the desired levels of throughput or reducing the storage overhead of the ledger. Elastico [26] proposed the first sharding-based permissionless blockchain, where the PoW is used for shard formation and the PBFT is run for intra-sharding consensus. Monoxide [27] is the first sharding mechanism that adopts asynchronous consensus. In order to prevent participants from aggregating mining powers in a shard, it allows participants to solve PoW puzzles at different shards concurrently. The ledger of SharPer [28] is formed as a DAG (Directed Acyclic Graph), and each cluster maintains only a view of the ledger. The decentralized flattened protocol is proposed to enable the parallel processing of cross-shard transactions with involved clusters.

Reputation promotes the robustness and efficiency of the sharding-based blockchain as well as provides incentives. The reputation can be evaluated based on the consensus activities of nodes [29–32] or ratings of other nodes [33,34]. Huang et al. [29] presented the double-chain architecture. It generates the reputation chain via the synchronous BFT consensus combining collective signing to support the high throughput of the transaction chain generated via the RAFT. To avoid collusive attacks, each shard should have similar

total reputation values. Wang et al. [30] integrated the RAFT with the Proof of Behavior (PoB) and supervisory mechanism for fast blockchains in complex networks. The PoB facilitates the election of a leader from honest nodes, and the supervisory group monitors whether a leader is acting honestly or not during a RAFT process. In [31], the trust concept is integrated into the deep Q network, which optimizes the system throughput and security level. The trust estimates the network's malicious probability by monitoring the inconsistency of block consensus. Zhang et al. [32] proposed the leader to be selected based on reputation values, several monitors in each committee to supervise the leader's behaviors, a referee committee to arbitrate between opposing parties, and a recovery procedure for the reselection of a leader. It mitigates a major bottleneck of the system via improving the capability of a leader. Gao et al. [33] adopted the sharding–hashgraph consensus mechanism to improve the throughput of blockchain networks. The weight of a node is evaluated according to node status, i.e., its geographic location, reputation values, etc. With the assistance of the weights, nodes are allocated to multiple shards with good performance, and their virtual voting enables efficient consensus. Asheralieva et al. [34] modeled a self-organized shard formation as the reputation-based coalitional game, where a node selects the best shard to maximize its payoff, and a shard coalition selects the best nodes to maximize its reputation. However, the above work pays less attention to inter-sharding consensus and shard reconfiguration, which are major features of the sharding-based blockchain.

2.2. MANET-Based Blockchain

The consensus mechanism of a MANET-based blockchain needs to satisfy the highly dynamic nature of nodes. According to the structure of the ledger, the MANET-based blockchains are classified into the chain and the DAG.

Some works utilize the chain structure for MANET-based blockchains. In order to incentivize mobile devices to share their computational resources, Rasool et al. [11] introduced a blockchain-based credit system by using Iroha, which is a project of Hyperledger for lightweight mobile applications. It also realizes reliable data analysis against malicious devices. Jiao et al. [12] proposed a stability-aware PoW consensus protocol, where the node with high stability is first selected as a miner. The stability of a node indicates its mobility in the network, which is calculated by the degree of the node in certain time slots. Liu et al. [10] introduced a blockchain system for security data collection in MANETs. In order to avoid forks incurred by network latency, after receiving the first valid block, a node stops mining and waits a time window for more valid blocks. When the time window expires, the winner is selected according to the specified rule, i.e., block generation time and the number of receipts within the block. Long et al. [15] optimized the block propagation in wireless blockchain networks by the partial engagement of nodes and optimal transmission routing based on a distributed Steiner tree. However, since the delay of transaction collection and block propagation is relatively high in MANETs [35], the geographical location of the miner and its connectivity in the network largely affect the blockchain performance.

Some works utilize the DAG structure for fast MANET-based blockchains. Morales et al. [36] integrated a blockgraph with RAFT to cope with network partition problems. The structure of the blockgraph contains multiple branches, and each branch corresponds to a chain of blocks generated by a network partition. When the network topology of a split, a merge or a mix occurs, the involved nodes process reconfiguration, leader elections and chain updates via the RAFT in different partitions. Luo et al. [13] presented a hashgraph-based solution for common knowledge formation in decentralized swarm robots. Swarm robots within a group reach consensus via hashgraph, and the ranger robot with powerful capabilities in communication and mobility establishes the connection among groups. Fu et al. [37] used hashgraph to support the dynamic changing of consensus subjects and proposed the TEE (Trusted Execution Environment)-based 'single-use of self-parent' mechanism to prevent fork attacks in hashgraph consensus. However, the above blockchains are vulnerable

to targeted attacks. For example, a partitioned network [36], a ranger robot [13] or the committee of consensus subjects dynamically joining or exiting [37] becomes the target.

Our work is improved from that of Zhang et al. [23] and Wang et al. [24], which alleviate targeted attacks incurred by nodes dynamically joining and leaving a shard. Howevaer, they are faced with the centralized problems of one miner synchronizing blocks to other nodes and a third party managing the reputation. Also, their infrastructure design cannot be adopted in MANETs.

3. Preliminary

Hashgraph [16] is a promising consensus mechanism based on DAG, realizing asynchronous byzantine fault tolerance. The decentralization feature of DAG enables hashgraph to work in a leaderless, asynchronous and parallel manner. Additionally, hashgraph achieves consensus on the order of transactions through processes: events generation, gossip about gossip, votes counting and ordering events.

Some concepts and operations of hashgraph are defined:

Event: An event is a data block, whose structure is denoted as (1). $timestamp$ is the generation time of the event, $hashId$ is the hash value of the event, and $txList$ is the transactions to be updated. $H1$ and $H2$ are the hash values of the node's previous event and the received event, respectively.

$$< timestamp, hashId, txList, H1, H2 > \tag{1}$$

Gossip about gossip: The event is sent to another node along with received gossip information and its own previous event.

Absolute Majority: More than $2/3$ of all nodes in the network.

Strongly Seeing: Event y strongly sees event x if the paths of y seeing x cover the absolute majority. For example, from the blue lines in Figure 2, event $b1$ is strongly visible to event $c3$, because $c3$ finds events $a2$, $b2$, $c2$ and $d2$ seeing $b1$.

Created Round: A newly created round starts from event y, when y strongly sees the events generated by the absolute majority. Event y is named as a witness. For example, in Figure 2, event $c3$ strongly sees events $b1$, $c1$, $d1$ and $e1$, so event $c3$ creates a new round.

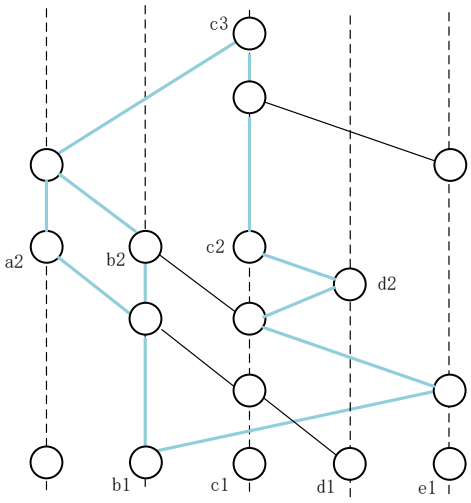


Figure 2. Illustration of strongly seeing in hashgraph.

Famous Witness: x is a famous witness if both x and z are witnesses, and z strongly sees more than $2/3$ of witnesses, each of which also strongly sees x. Famous witnesses are confirmed by the community, which is proved by [16].

Event Order: Famous witnesses are sorted according to their received round number, consensus time and signature XORing. The sequence is consistent among honest nodes. It means that the order cannot be tampered with unless it is approved by the absolute majority.

4. Behavior-Based Sharding Hashgraph Framework

In this section, the system model, network model and security problems are introduced.

4.1. System Model

The shards are associated with geographical regions. Disjoint mobile nodes within each shard maintain a graph via hashgraph consensus in Figure 3. The regions of shards partially overlap to increase the communication channels among shards. During the consensus process, dishonest behaviors of nodes are updated to the graph and become a decentralized blacklist.

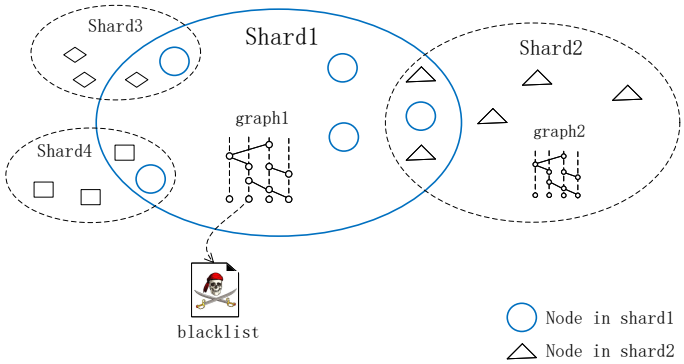


Figure 3. The system model of sharding hashgraph.

In the consensus layer of hashgraph, three dishonest behaviors are defined as a *False Event (FE)*, *False Free-riding Event (FFE)* and *Double-spent Event (DE)*. The False Event has invalid transactions in its $txList$. The False Free-riding Event is the event referring to False Events, meaning that the event generator performs false verification. The Double-spent Events are defined by (2). If only one of the Double-spent Events becomes famous, a fork occurs. Proved by [20], at least one honest node can see Double-spent Events. These events can be found by honest nodes and added into an accusation transaction in (3), where pk is the public key of the accused node, which is assumed to be its ID. $bType$ is the type of dishonest behavior, and beH is the hash value of the False Event or False Free-riding Event, or hash values of Double-spent Events. Thus, honest nodes validate the events with tx^a in the following rounds of consensus. tx^as form the decentralized blacklist of nodes, recording their dishonest behaviors in consensus.

$$\begin{aligned} &< event_i, event_j > \\ s.t. &\exists x \in event_i.txList, \exists y \in event_j.txList \\ &x \text{ and } y \text{ are conflicting} \\ &event_i \text{ and } event_j \text{ are generated by the same node} \end{aligned} \tag{2}$$

$$\begin{aligned} tx^a &= \{pk, beH, bType, timestamp\} \\ s.t. bType &= FE|FFE|DE \end{aligned} \tag{3}$$

In the transmission layer of hashgraph, nodes are required to propagate received gossip information. Two propagation behaviors of nodes are defined as a *Empty Event* and *Non-empty Event*. An Empty Event is the event where $txList = null$. A Non-empty Event is the event where $txList \neq null$, meaning the involved transactions are also generated in the

event. First, according to a given time scale, e.g., a week, the Empty Events and Non-empty Events are divided into recent events and past events in the graph. Second, the network connectivity is formally described as a vector $\{eNum, gNum\}$, where $eNum$ is the number of generated events and $gNum$ is the number of event generators. According to a given standard network connectivity, $\{\delta, \Delta\}$, if $eNum < \delta$ & $gNum < \Delta$, the network connectivity is poor. If $eNum > \delta$ & $gNum > \Delta$, the network connectivity is good. Otherwise, if $eNum \leq \delta \,||\, gNum \geq \Delta$, or $eNum \geq \delta \,||\, gNum \leq \Delta$, the network connectivity is normal. Third, denote the nodes without transactions in the blacklist as $hNodes$. $hNodes$ have not engaged in dishonest behaviors. According to the graph, the Empty Events generated by $hNodes$ are counted as eN_1, eN_2, eN_3, eN_4, eN_5 and eN_6, and the Non-empty Events generated by $hNodes$ are counted as nN_1, nN_2, nN_3, nN_4, nN_5 and nN_6 in Table 1. For example, in the past network with good connectivity, the node generated 5 Empty Events and 11 Non-empty Events. In the recent network with poor connectivity, the node also generated 1 Empty Events and 11 Non-empty Events. So, for this node, $eN_5 = 5$, $nN_5 = 15$, $eN_1 = 1$, $nN_1 = 11$, and other parameters are 0. Thus, Table 1 counts the honest propagation behaviors of $hNodes$.

Table 1. The classification of honest Empty Events and Non-empty Events based on their generation time and network connectivity.

	Poor Network Connectivity	**Good Network Connectivity**	**Normal Network Connectivity**
Recent events	eN_1, nN_1	eN_2, nN_2	eN_3, nN_3
Past events	eN_4, nN_4	eN_5, nN_5	eN_6, nN_6

According to Table 1, the propagation behaviors of a candidate receiver and a candidate sender are evaluated. On one hand, provided that a node intends to send an event to one of its neighbors, in order to select a reliable neighbor to forward the event in time, the reputation value of each neighbor (candidate receiver) is calculated in a weighted Formula (4). α_1 and α_2 represent the weights of recent events and past events, respectively. $\alpha_1 + \alpha_2 = 1$ and $\alpha_1 > \alpha_2$ make recent events more influential. Events will be sent to the receiver with high $rRep$. On the other hand, provided that a node receives requests from its multiple neighbors, it needs to choose one of them as a sender. In order to receive valid events and improve communication efficiency in the network, the reputation value of each neighbor (candidate sender) is calculated in (5). β is a given value to control the weight of a Empty Event and Non-empty Event, $\beta \in (0.5, 1]$. Since an Empty Event is only to relay gossip information, its generation should be associated with the network connectivity. Particularly, when the network connectivity is poor, e.g., nodes moving frequently, more Empty Events are conducive to event propagation. So, the weight of an Empty Event is β, while that of a Non-empty Event is $(1 - \beta)$. When the network connectivity is good, fewer Empty Events can cut down the number of repeated events transmitted, mitigating the channel competition and reducing network resource overhead. So, the weight of an Empty Event is $(1 - \beta)$, while that of a Non-empty Event is β. Events from the sender with a high $sRep$ will be received. Note that in order to broadcast an event, the sender and receiver of a node are different.

$$rRep = \alpha_1 * (eN_1 + eN_2 + eN_3 + nN_1 + nN_2 + nN_3) + \alpha_2 * (eN_4 + eN_5 + eN_6 + nN_4 + nN_5 + nN_6) \tag{4}$$

$$\begin{aligned} sRep = {} & \alpha_1 * [(\beta \cdot eN_1 + (1-\beta) \cdot nN_1) + (\beta \cdot nN_2 + (1-\beta) \cdot eN_2) + (0.5 \cdot eN_3 + 0.5 \cdot nN_3)] \\ & + \alpha_2 * [(\beta \cdot eN_4 + (1-\beta_1) \cdot nN_4) + (\beta \cdot nN_5 + (1-\beta) \cdot eN_5) + (0.5 \cdot eN_6 + 0.5 \cdot nN_6)] \end{aligned} \tag{5}$$

Therefore, the hashgraph mechanism with a blacklist brings some benefits: (i) the nodes with dishonest behaviors are found and can be punished in a decentralized manner, e.g., the node appearing in the blacklist frequently is not allowed to participate in a shard. The accused behaviors are traceable and auditable in case of a dispute. Rational nodes are

encouraged to work honestly. (ii) Nodes intend to transmit events continuously, because the nodes with smaller *rRep* or *sRep* cannot receive events or send events without difficulty. The problem of dishonest gossip, that the gossip information contains False Events, False Free-riding Events and Double-spent Events, is alleviated. It reduces the transmission delay of events and improves the efficiency of hashgraph consensus.

4.2. Network Model

The network model is built on the mobile ad hoc network, which is also called a wireless multi-hop network without infrastructures. In the network, neighbor shards are two shards that have at least one pair of neighbor nodes. Since nodes within a shard are in geographical proximity, there are more neighbor nodes and multiple communication paths between every two nodes. Hence, the transmission delay in a shard is less. Nevertheless, due to opportunistic communications caused by node mobility, neighbor nodes change over time, leading to changes of neighbor shards. Hence, the connection channels between shards are not always reliable, and the communication among them has a certain delay.

4.3. Security Problem

However, the dynamic shards caused by nodes dynamically joining or exiting incur some security problems.

- Sybil attack. A malicious node intends to create multiple identities to participate in shards, or an identity is used in multiple shards. Intuitively, only when the identity is validated by multiple shards is the node allowed to join a shard. Nevertheless, under unstable communications among shards, high transmission delay makes the node spend a long time waiting for identity authentication.
- Targeted attack. Malicious nodes intend to control a specified shard by compromising honest nodes or joining the shard. When the number of malicious nodes exceeds the fault tolerance of the shard, it is corrupted. Intuitively, the reconfiguration of shards in a small epoch can alleviate the targeted attack. Nevertheless, it results in the frequent interruption of services during shard reconfiguration, reducing the performance of the system.

5. Asynchronous Intra-Sharding and Inter-Sharding Consensus

In this section, the principle of tree-assisted inter-sharding consensus and the work process of a dynamic shard are discussed.

5.1. Principle of Tree-Assisted Inter-Sharding Consensus

The basic requirement of inter-sharding consensus is to guarantee the consistency and atomicity of a cross-shard transaction. The principle of the tree-assisted inter-sharding consensus is that cross-shard transactions are updated first and then confirmed so that eventual consistency is achieved. Since a tree is proposed to include the cross-shard transaction and its related transactions, the atomicity of the cross-shard transaction can also be guaranteed. The details of this principle are described as follows.

- A cross-shard transaction is generated and updated to the graph of the local shard and sent to involved shards.
- Transactions related to the cross-shard transaction are generated and updated by involved shards in parallel. They refer to the cross-shard transaction and are sent to other involved shards, constructing the tree globally.
- When the cross-shard transaction is used, the number of child nodes of the cross-shard transaction and their validity are checked. Only if the cross-shard transaction is validated by all involved shards is it confirmed. Otherwise, if a given time is expired, the cross-shard transaction is revoked by a newly updated transaction, and the event containing the cross-shard transaction is determined to be a False Event and added into the blacklist.

Thus, the efficiency of updating valid cross-shard transactions is improved. Its update time is only related to the delay of consensus in a local shard. The inter-sharding consensus is suitable to the MANET, since the communication between shards is opportunistic. In addition, the generation of cross-shard transactions is accountable.

5.2. Work Process of a Dynamic Shard

A dynamic shard is incurred by the addition and deletion of nodes. In addition to the tree-assisted inter-sharding consensus, hashgraph consensus is also used to update transactions first before confirming them. The asynchronous intra-sharding and inter-sharding consensus empower the security and efficiency of a dynamic shard via four steps:

Step 1: Request of an accounting node.

When a node enters the geographical range of a shard, it can request to become an accounting node by submitting a *cross-shard joined transaction* in (6). $sig()$ is a signature function, sk is the secret key of the requester, and ts is the generation time of cTx^j. Denote the shard in this application as *target shard* whose ID is sId. H is the hash value of cTx^j, and $preH$ is the hash value of the requester's previous valid transaction.

$$cTx^j = sig(sk, ts, sId, H, preH, beSet) \tag{6}$$

$$s.t. beSet = \{be_1, be_2, be_3, ...\} \tag{7}$$

$$be =< eH, preH, jTs, eTs, jSId, nBL >$$

In (7), each element in $beSet$ records the historical behaviors of the requester after given the timestamp ts_0. Since these records are only updated when cTx^j is approved, each be has a relevant pair of cTx^j and the *cross-shard exited transaction*. jTs and eTs are the generation time of cTx^j and the *cross-shard exited transaction*, respectively. $eTs > jTs > ts_0$. $jSId$ is the ID of the shard related to the behaivors be, and nBL is the number of the accusation transactions with $timestamp \in [jTs, eTs]$. eH is the hash value of the *cross-shard exited transaction*, and $preH = cTx^j.preH$.

Nodes in *target shard* check the records form a hash link to prevent the behavior records from decoupling, which is explained in the Performance Evaluation section. If cTx^j is approved, it is sent to the involved shards.

Meanwhile, the requester downloads validated events and shard information in *target shard*. For example, shard information contains the geographical range and the member IDs of the shard. It acts as a relay to propagate gossip information via events with $txList = null$, and it generates transactions for data sharing. It is worthwhile to note that these transactions can not be used until cTx^j is confirmed. Behaviors of the requester are accountable, because the False Event of cTx^j and False Free-riding Events can be recorded in the blacklist.

Step 2: 1st validation of the request. Nodes in involved shards check whether the behavior records of cTx^j are correct.

In an involved shard, cTx^j is verified via the smart contract (see Algorithm 1). In line 1, the signature of the requester, and whether the requester is working in a shard, are checked. In line 4, $|beS|$ is the number of elements in beS, and $|eSet|$ is the number of elements in $eSet$. In lines 5–7, according to the graph, behavior records in beS are checked. Particularly, for each be, if the *cross-shard exited transaction* does not exist, or the hash value of the previous *cross-shard joined transaction*, the joined timestamp, the exited timestamp, or the number of accusation transactions is incorrect, False is returned. In line 8, joined transaction tx^j is generated to validate cTx^j, where H is the hash value of tx^j, $jH = cTx^j.H$, $ISId$ is the ID of the involved shard, sId is the ID of the *target shard*, $sId = cTx^j.sId$, and ts is the generation time of tx^j.

Algorithm 1 cTx^j is verified in an involved shard.

Input: cTx^j, the graph in the involved shard
Output: tx^j or False

1: **if** the signature of the requester is valid && there is a latest *cross-shard joined transaction* before a latest *cross-shard exited transaction* of the requester **then**
2: Get $beS = \{be | be \in cTx^j.beSet$ && $be.jSId$ is the ID of the shard$\}$
3: Retrieve the set of requester's *cross-shard joined transactions*, *jSet*, and set of requester's *cross-shard existed transactions*, *eSet*, from local graph
4: **if** $|beS| = |eSet|$ **then**
5: **for** each *be* in *beS* **do**
6: Get an element cTx'^e which $H = be.eH$ from *eSet*
7: Get an element cTx'^j which $H = cTx'^e.preH$ from *jSet*
8: **if** cTx'^e not exist || $cTx'^j.preH \neq be.preH$ || $cTx'^j.ts \neq be.jTs$ || $cTx'^e.ts \neq be.eTs$ || $be.nBl$ is wrong **then return** False
9: Generate $tx^j = < H, jH, ISId, sId, ts >$
10: **else return** False
11: **else return** False

Then, tx^j is added to an event, updated to the graph, and sent to other involved shards and *target shard*. In this way, a two-layer tree, $tree^{je}$, is constructed in these shards. For example, in Figure 4, $tree^{je}$ is composed of cTx^j, tx_2^j and tx_3^j. cTx^j is the root, and tx_2^j and tx_3^j are the first validation of cTx^j.

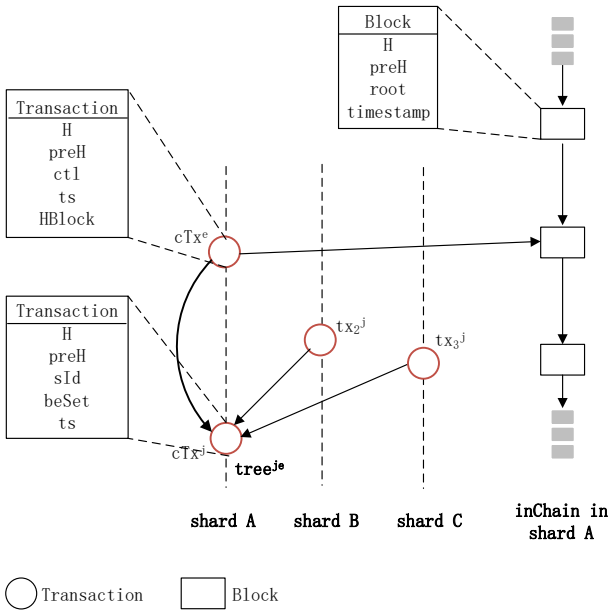


Figure 4. The structure of $tree^{je}$ and *inChain*.

Step 3: 2nd validation of the request. Nodes in the *target shard* check whether the reputation values of the requester are greater than a given threshold.

In the *target shard*, when the unconfirmed transactions of the requester are used, the second verification of cTx^j is processed via the smart contract (see Algorithm 2). In line 8, the reputation value of a requester is calculated via a sigmoid function. α is set to larger than 1 in order to make the weight of recent behaviors on reputation evaluation greater than that of the past. λ and ϵ are preset values. If *rep* of the requester is more than threshold $th1$, the requester becomes an accounting node of the *target shard*.

Otherwise, if a given time is expired, a *cross-shard exited transaction* named cTx^e is generated in line 11, where H is the hash value of cTx^e, $preH = cTx^j.H$, ts is the generation

time of cTx^e, and $HBlock = null$, which is discussed later. ctl is a list of the requester's generated transactions since cTx^j is submitted, which will be invalid. cTx^e is sent to involved shards. In addition, tx^a is generated in line 12 and added into the blacklist of the *target shard*.

Algorithm 2 cTx^j is verified in *target shard*.

Input: $tree^{je}$
Output: *True* , or (cTx^e, tx^a)

1: Get the root of $tree^{je}$, cTx^j
2: Get children of cTx^j in $tree^{je}$, $cSet$
3: Retrieve IDs $idSet$ of involved shards from $cTx^j.beSet$
4: **if** $\forall id \in idSet$ && $id = cSet.tx^j.ISId$ **then**
5: Sequence elements in $cTx^j.beSet$ according to $be_i.preH = be_k.eH$ and be_k as the previous one of be_i
6: **for** Each be in $cTx^j.beSet$ in ascending order **do**
7: $duration = be.eTs - be.jTs$
8: $rep = \alpha \cdot rep + \frac{(be.jTs+be.eTs)/2}{1+e^{-\lambda \cdot (duration/be.nBL-\epsilon)}}$
9: **if** $rep > th1$ **then return** *True*
10: **if** a given time is expired **then**
11: Generate $cTx^e = < H, preH, ctl, ts, HBlock >$
12: Generate tx^a with pk as the public key of the request & $beH = cTx^j.H$ & $bType = FE$ according to (3)

Red vertexes in Figure 4 present the tree structure of root cTx^j in inter-sharding consensus. After cTx^j is updated to the graph in shard A, it is sent to involved shards, i.e., shards B and C. Nodes in the involved shards verify the behavior records of the requester in parallel (see Algorithm 1). If the records are correct, involved shards send tx_2^j and tx_3^j to shard A, referring to cTx^j. In shard A, only if all involved shards validate cTx^j and the reputation value of the node is larger than threshold $th1$ are the behavior records in cTx^j valid; otherwise, cTx^e is generated, referring to cTx^j (see Algorithm 2). Hence, only two rounds of intra-sharding consensus are required to validate behavior records. In addition, when cTx^j is validated, the requester's previous votes in gossip are counted in the graph. In this way, the requester does not need to wait for its consensus activities, and its updated transactions can be used directly. Hence, the efficiency of a requester participating in a shard is improved.

Step 4: The requester leaves a shard.

In hashgraph consensus, a chain named *inChain* is built for event retrieval and shard security. Confirmed events are ordered and packaged into a block, and then the block is updated to *inChain* in every preset period, e.g., 10 received rounds of hashgraph or 10 min. The block structure in *inChain* is $< H, preH, root, timestamp >$. Specifically, H is the hash value of the block, $preH$ refers to the previous valid block, $root$ is a root of the Merkel Hash Tree whose leaves are confirmed events, and $timestamp$ is the generation time of the block. For example, shard A's *inChain* is depicted in Figure 4.

When an accounting node leaves a shard normally, cTx^e is generated in (8), where $ctl = null$ and $HBlock$ refer to the latest block of *inChain*. Once cTx^e is confirmed in *target shard*, it is sent to involved shards.

$$cTx^e =< H, preH, ctl, ts, HBlock > \tag{8}$$

6. Shard Recovery and Reconfiguration

With the assistance of cTx^e, the compromised shard can be recovered by other shards. To help readers clearly understand the scheme, suppose that at most, one shard is corrupted in a time. When the shard is found to be corrupted, a node within it sends a request of shard

recovery to two shards along with its local *inChain* denoted by $chainA'$. The latest trusted block of *inChain* is denoted by $ltBlock$ and determined based on the majority principle. For example, shards D and E are requested to recover shard A, because the two latest cTx^es generated from shard A are kept in shards D and E. According to cTx^es, the latest blocks of $chainA'$ in shards D and E are $D1$ and $E1$, respectively. Shard A confirms the latest block in $chainA'$ as $A1$. $ltBlock$ is generated in three cases in Figure 5. Particularly, $ltBlock = A1$ for case1, $ltBlock = E1$ for case2, and $ltBlock = D1$ for case3. In case2, shard A is determined to be corrupted, because it does not agree with $E1$ confirmed by the other two shards.

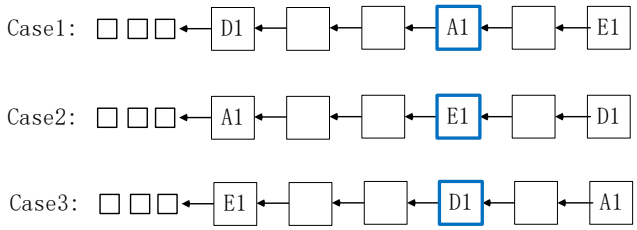


Figure 5. Three cases of a latest block in $chainA'$ determined by the majority principle of shards A, D and E.

In case1 or case3, the reputation evaluation of shard A is based on the changes in honest/dishonest behaviors of nodes over time. p indicates the period between the time of the last sharding configuration and the timestamp of $ltBlock$. According to the events with $timestamp \in p$, the reputation values of shard A are calculated via a sigmoid function in (9), where λ and ϵ as preset values control the range of the fast growth interval of the formula. I is the node whose events are included in the blacklist, and SD is the standard deviation of honest/dishonest behaviors of the node. In (10), p is divided into T time slots. In each time slot, r is the ratio of the number of times the node's events are included in the blacklist ($nNBL$) and the number of confirmed events ($nEvent$). u is the average of r in T.

$$shRep^p = \frac{1}{1 + e^{-\lambda \cdot (1/\sum_{i \in I} SD_i - \epsilon)}} \tag{9}$$

$$\begin{aligned} s.t. SD &= \sqrt{\frac{\sum_{t=1}^{T}(r_t - u)^2}{T}} \\ r &= nNBL/nEvent \\ u &= \sum_{t=1}^{T} r_t / T \end{aligned} \tag{10}$$

For example, if a list of r of the node is $setR1 = \{0, 0, 0.5, 0.5\}$, the node is more likely to have been compromised, because its generated events start to be included in the blacklist in the last two time slots. If a list of r of the node is $setR2 = \{0.5, 0.5, 0, 0\}$, the node is more likely to launch on/off attacks, because it starts to work honestly or does not have any behaviors in the last two time slots. In Figure 6, the reputation values of shards whose nodes have $setR1$, $setR2$ and $setR3 = \{0.1, 0.2, 0.3, 0.4\}$ are presented. The reputation values of the first two shards are the same, which are smaller than that of the third shard. It is because the changes of nodes' honest/dishonest behaviors in the first two shards are more frequent. In other words, they have more potential security risks. In addition, the reputation value of the shard decreases when more nodes in the shard change honest/dishonest behaviors.

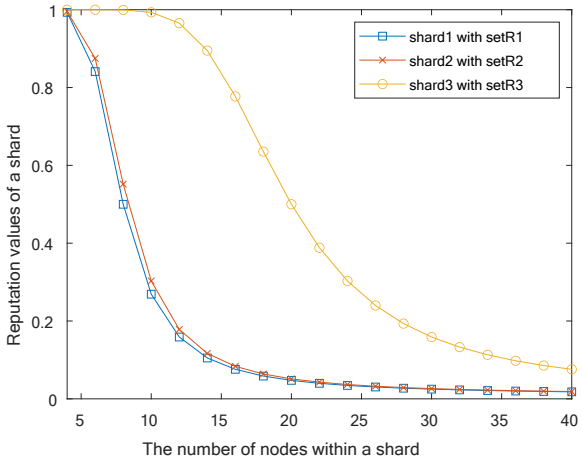


Figure 6. According to (9), $\lambda = 10$ and $\epsilon = 0.5$. The lists of r of nodes in shard1, shard2 and shard3 are $\{0, 0, 0.5, 0.5\}$, $\{0.5, 0.5, 0, 0\}$ and $\{0.1, 0.2, 0.3, 0.4\}$, respectively. Hence, $T = 4$ and $u = 0.25$. The reputation values of shard1, shard2 and shard3 are presented with the increase of nodes in the shards.

Denote the chain in shard A as $chainA$. When the reputation value of shard A is less than threshold $th2$, shard D and E replay the transactions after the latest trusted block in $chainA$. If a false transaction is found by both shard D and E, shard A is determined to be corrupted. Then, shard A is reconfigured, and the latest block of the chain is set to $ltBlock$. In addition, accusation transactions associated to False Events, False Free-riding Events or Double-spent Events in $chainA$ are generated and added into the blacklist of shard A. Thus, as long as one node within a shard is honest, the shard can be recovered and its security is guaranteed.

The algorithm of shard recovery is presented in Algorithm 3. The input is cTx^es, $chainA'$ is provided by a requester, and $chainA$ is retrieved from shard A.

Algorithm 3 Recovery of Shard A by shard D and E.

Input: cTx^es, $chainA'$, $chainA$
Output: Recovery of Shard A

For a node within shard A
The node sends $chainA'$ to shard D and E to accuse shard A of being corrupted

For shards A, D and E
According to cTx^es, the lastest trusted block, $ltBlock$, is confirmed through the majority principle

For both shards D and E
if $ltBlock$ is agreed by shard A **then**
    Calculate the reputation value of shard A, $shRep$, according to (9)
    if $shRep < th2$ **then**
        Verify transactions after $ltBlock$ in $chainA$
        if any false transaction found in $chainA$ **then**
            Shard A is reconfigured
            $ltBlock$ is the latest block of the chain in shard A
else
    Shard A is reconfigured
    $ltBlock$ is the latest block of the chain in shard A

Moreover, an adaptive shard reconfiguration is devised. Compared to a fixed epoch widely used in traditional shard reconfiguration, e.g., Elastico [26] and Monoxide [27], the dynamic shard reconfiguration is based on the shard reputation in (9). That is, when the reputation value of a shard is less than $th3$, $the3 < th2$, the shard is reconfigured by the shuffling of nodes. The nodes with more events included in the blacklist are removed from the shard.

7. Performance Evaluation

In this section, security analysis and experimental evaluation are discussed.

7.1. Security Analysis

The attack prevention and properties of the proposed scheme are analyzed as follows.

- Sybil attack

The Sybil attack is resisted through the historical behavior records of nodes and inter-sharding consensus. The node applies to join a shard by submitting its historical behavior records. Only if the records are validated by all involved shards, and the reputation value calculated from these records is greater than threshold $th1$, is the node allowed to participate in the consensus of the shard and data sharing in the shard. $th1$ can be $\{rep|rep.rank = 2/3\}$. It means when the reputation values of nodes within the shard are sorted in descending order, rep is ranked as $2/3$ of all values.

Theorem 1. *The behavior records of the node in involved shards cannot be reused to generate multiple identities.*

Proof. In line 11 of Algorithm 2 and (8), *a cross-shard exited transaction*, cTx^e, is generated, where $preH = cTx^j.H$. cTx^e refers to cTx^j. cTx^j and cTx^e are sent to involved shards. It means when the node is removed from a shard, its cTx^e is updated to the graphs of all involved shards, referring to cTx^j. In other words, if there is not a latest *cross-shard exited transaction* after a latest *cross-shard joined transaction* of the requester in involved shards, the requester is working in a shard. The state of the requester becomes *working*, and honest nodes are aware of its state. Its behavior records are locked and cannot be used as sharding requests. □

Theorem 2. *It is hard to decouple the behavior records of the requester to generate multiple identities or launch whitewashing attacks.*

Proof. First, the behavior records of the requester cannot be decoupled into discontinuous parts. For a node, its $cTx^j.preH$ indicates the hash value of its previous *cross-shard exited transaction*. The two transactions belong to different behavior records and may be recorded in graphs of different shards. The chain of records is defined as (11), where *beSet* is the behavior records of the request, be_k is the previous behavior record of be_i, $be_i.preH = cTx_i^j.preH$, and $be_k.eH = cTx_k^e.H$. Each record has a relevant pair of *cross-shard joined transactions* and *cross-shard exited transactions*. Hence, the behavior records are in the chain structure depicted in Figure 7. In (12), be_k is determined to be the previous behavior record of be_i, where $be.eTs = cTx^e.ts$ and be_0 as a genesis record. Nodes in the *target shard* check whether the behavior records in the application meet (11). If it does not meet the requirement, the participating application is not approved.

$$\{be_i|\forall be_i \in beSet \;\&\; be_i.preH = be_k.eH\} \tag{11}$$

$$s.t.\{be_k|(be_k \in beSet' \;\&\; be_k.eTs = max(beSet'.be.eTs)) \;||\; be_k = be_0\} \tag{12}$$

$$\{beSet'|beSet' \subset beSet \;\&\; beSet'.be.eTs < be_i.eTs\}$$

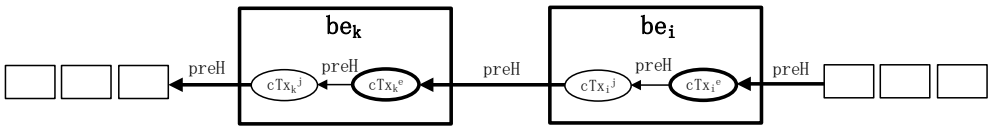


Figure 7. The chain structure of behavior records and transactions.

Second, the records of the requester within a shard can not be decoupled. The behavior records that are valid and include all records of the requester in the shard are defined as (13), where CTX^e and CTX^j are the set of *cross-shard exited transactions* and *cross-shard joined transactions* of the requester in the shard. $f1$ means that $beSet_v.be.nBl$ is the number of the accusation transactions with $timestamp \in [beSet_v.be.jTs, beSet_v.be.eTs]$. Nodes in an involved shard check whether the behavior records in the application meet (13). If they do not meet the requirement, the participating application is not approved.

$$\begin{aligned}\{beSet_v | beSet_v \subseteq beSet \;\&\; \forall cTx^e \in CTX^e \;\&\; beSet_v.be.eH &= cTx^e.H \;\&\\ beSet_v.be.eTs &= cTx^e.ts \;\&\; cTx^j \in CTX^j \;\&\; cTx^j.H = cTx^e.preH \;\&\\ beSet_v.be.jTs &= cTx^j.ts \;\&\; f1\}\end{aligned} \tag{13}$$

Third, the behavior records of the requester cannot be decoupled to create multiple identities involving one shard. Different applications uploaded by the requester are defined as $cTx_k^j.sign = cTx_g^j.sign$, where cTx_k^j and cTx_g^j are current applications, $cTx_k^j \neq cTx_g^j$, and $sign$ is the signature of the generator. Nodes in a shard check the applications. If they meet the definition, the participating applications are not approved. □

- Targeted attack

The targeted attack related to shards is resisted through shard recovery and reconfiguration. First, when the malicious nodes exceed 2/3 of all nodes, if a shard is being corrupted, as long as there is one honest node in the shard, the shard can be restored by other shards. In addition, in order to prevent malicious nodes from frequently accusing a shard and consuming its available resources, shard recovery proceeds only if the reputation value of the shard is less than threshold $th2$. Second, when the reputation value of a shard is less than the threshold $th3$, $th3 < th2$, the shard is reconfigured. $th2$ and $th3$ can be proportional to $avgHF$, $avgSR$ and fre. $avgHF$ is the average failure rate of hardware deployed in nodes. $avgSR$ is the average of the smallest x reputation values of shards, e.g., $x = 4$. fre is the frequency of nodes entering and leaving the shard.

The security assumption of targeted attacks is a reliable reputation evaluation of a shard. The reputation value of a shard is calculated based on the changes of honest/dishonest behaviors of nodes within it. Thus, we assume the following: (i) when the shard is configured, the number of malicious nodes is less than 1/3 of all nodes. (ii) During the process of a shard being compromised, the honest nodes start to work dishonestly, or the malicious nodes in turn work honestly.

- Accountability

Nodes are accountable for their consensus behaviors by using a decentralized blacklist. The dishonest behaviors related to a False Event, False Free-riding Event and Double-spent Event of nodes can be found in the graph and updated in the blacklist. With the assistance of the blacklist, the honest gossip behaviors of the node are evaluated, and its reputation values as a candidate sender, $sRep$, and reputation values as a candidate receiver, $rRep$, are calculated. Events generated from the neighbor nodes with high $sRep$ are received first, and events are sent to the neighbor nodes with high $rRep$ first. In this way, these nodes have more opportunities to benefit from data sharing. Furthermore, the nodes whose behaviors are included in the blacklist are more likely to be rejected for sharding. Particularly, the

node's request of joining a new shard is rejected, or it is removed from the shard in the shuffling of nodes.

- Adaptability

The shard is dynamically adaptive in terms of consensus nodes and reconfiguration epochs. On one hand, different from traditional hashgraph, nodes can dynamically join and leave the consensus via cross-shard joined and exited transactions. It means when the joined transaction is confirmed, honest nodes are aware of the participating node as well as counting its virtual votes for consensus. It improves the scalability of the consensus. On the other hand, different from a fixed epoch for shard reconfiguration, nodes of the shard are shuffled when the reputation value of the shard is less than a threshold. It avoids the problem of a larger epoch, which makes a shard vulnerable to being compromised, and a smaller epoch results in the frequent interruption of services during shard reconfiguration.

- Decentralization

The reputation model is decentralized, preventing a Single-Point Failure. The dishonest behaviors of nodes are recorded in a decentralized blacklist and utilized to generate reputation values of nodes or shards through smart contracts. Furthermore, our consensus is full decentralization because there is no need for a leader.

7.2. Experimental Evaluation

7.2.1. Experimental Setting

Opnet [38] is utilized to simulate the communication of consensus and disclose the effects of mobility on consensus. It is a popular network simulator of discrete events, providing a realistic and reliable simulation environment in most network types. Some models are pre-built in Opnet and easily deployed for MANET simulation, including a traffic model, statistical model, mobility config, failure config, routing protocol, etc.

The scenario map is 4 km $\times$ 4 km. Nodes are identified by their IP address, and they move in a random direction with a power of 0.05 dB and a speed of 50 m/s. The number of failure nodes is less than 1/3 of all nodes. Packets are propagated through the aodv routing protocol in the channel bandwidth of 11 Mb/s without being reassembled during transmission. The traffic is generated in an exponential distribution. Each simulation under the same condition is replicated ten times, and the average of these 10 data points is considered.

7.2.2. Experimental Results

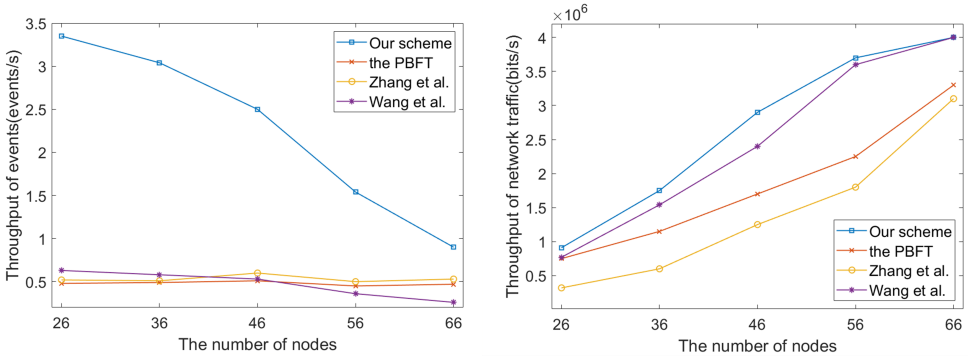
- Intra-sharding consensus

Table 2 presents the comparison of the PBFT, traditional hashgraph consensus, the consensus in Zhang et al. [23], the consensus in Wang et al. [24], and our proposed scheme. Consensus in [23] is based on PoW. The consensus process in [24] contains two rounds, the first round of which is realized by a primary and primary candidates based on hashgraph, and the second round of which is a primary synchronizing macro-block to accounting nodes. To make the explanation clear, blocks in the PBFT, refs. [23,24], are regarded as events.

Table 2. Comparison among popular consensus and our scheme in MANETs. $r1FT$ is 1/3 of a primary and primary candidates, and $r2FT$ is 1/2 of a primary and accounting nodes.

Consensus Scheme	Consistency	Leader-Based	Fault Tolerance	Neighbors Selection for Events Propagationn
PBFT	Strong	Leader	less than 1/3 of all nodes	Random
traditional hashgraph	Eventual	Leaderless	less than 1/3 of all nodes	Random
Zhang et al. [23]	Eventual	Leader	less than 1/2 of all nodes	Random
Wang et al. [24]	Eventual	Leader	less than min $(r1FT, r2FT)$	Random
our scheme	Eventual	Leaderless	less than 1/3 of all nodes	Selective

Figure 8 presents the changes of event throughput and traffic throughput with the increase of consensus nodes. In Figure 8a, our proposed scheme is better than other schemes, because five nodes generate events in parallel in our scheme, while only one miner/leader/primary generates events in the PBFT, as shown in Zhang et al. [23] and Wang et al. [24]. With the increase of consensus nodes, the throughput drops and the gap between our scheme and another scheme becomes smaller. The reason is that the increasing events generated by more nodes result in network congestion, especially events in our scheme growing faster. However, since our scheme adopts sharding technology, the number of nodes can be controlled effectively. In Figure 8b, traffic information includes events traffic, control traffic and routing traffic. The performance of our scheme is also better than that in other schemes. Although the traffic throughput of our scheme is very close to [24] when the number of nodes is 66, more events in our scheme are generated according to Figure 8a. In other words, less control traffic and routing traffic are consumed to be associated with event updates in our scheme, so our communication overhead is efficient. In addition, the traffic throughput of the PBFT is more than that of [23], while the event throughput of [23] is better. It is because more synchronous messages are generated in the PBFT, and these messages lead to a high delay of consensus in opportunistic communications. The PBFT is not very suitable for mobile consensus in terms of throughput.

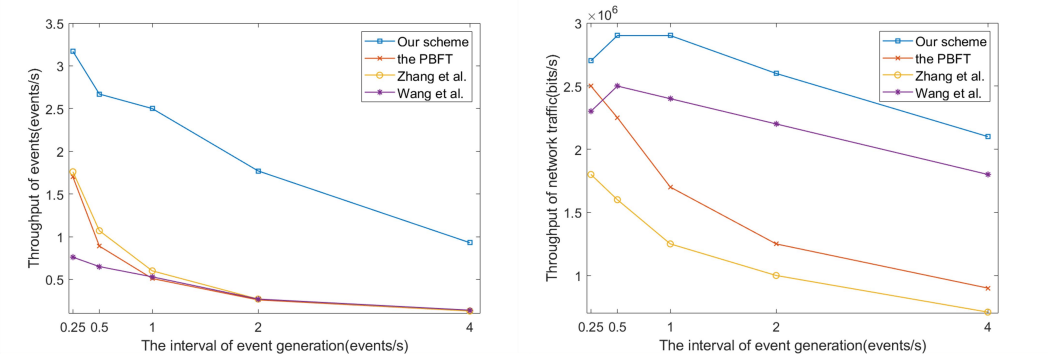


(a) The changes of event throughput with the increase of consensus nodes

(b) The changes of network traffic throughput with the increase of consensus nodes

Figure 8. The changes of event throughput and network traffic throughput with the increase of consensus nodes. For example, in our scheme, 46 nodes can achieve fault tolerance for 15 Byzantine nodes. Zhang et al. [23], Wang et al. [24].

Figure 9 presents the changes in event throughput and traffic throughput with the increase of event generation interval. In Figure 9a, the event throughput of our scheme is better than other schemes, because more than one node generates events in our scheme. After an interval of 1 s, the event throughput of Zhang et al. [23], Wang et al. [24] and the PBFT are very similar, because there are more available network resources and the same number of events can be updated in a similar amount of time. Furthermore, the gap of event throughput between our scheme and another scheme becomes smaller in Figure 9a, while the gap of traffic throughput between our scheme and another scheme remains stable after an interval of 2 s in Figure 9b. It is because our scheme uses gossip about gossip as a propagation method. When the interval of event generation increases, fewer gossip messages lead to more control traffic being generated. In other words, lower data rates can reduce the performance of our scheme. In addition, in Figure 9b, the gap between our scheme and [24] is stable, since both are based on hashgraph.

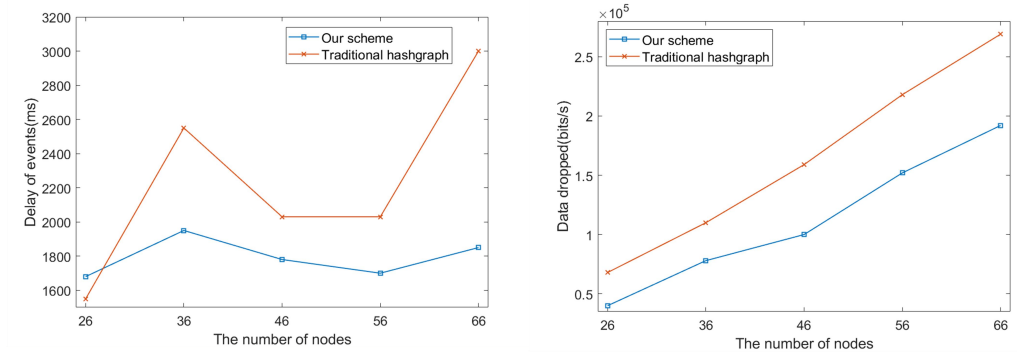


(a) The changes of event throughput with the increase of intervals of event generation

(b) The changes of network traffic throughput with the increase of intervals of event generation

Figure 9. The changes of event throughput and traffic throughput with the increase of intervals of event generation in the case of 46 nodes. Zhang et al. [23], Wang et al. [24].

Figure 10 presents the changes in event delay and data dropped rates with the increase of consensus nodes. In Figure 10a, with the increase of consensus nodes, the changes of delay are not monotonic. The reason is that the major factors affecting delay include the number of hops, network congestion, channel competition and the number of isolated nodes. The increase of nodes has a negative impact on the delay via the first three factors, while it has a positive impact on the delay via the fourth factor. The delay of our scheme is better than that of traditional hashgraph, because our scheme selects honest neighbors to propagate events, and fewer transmitted events mitigate the competition of wireless channels. For a similar reason, the data dropped rates of our scheme are less than those of the traditional hashgraph in Figure 10b. In Figure 10b, with the increase of consensus nodes, data dropped rates increase as well. It is because more data packets are generated to achieve consensus on an event, and some of them are dropped in opportunistic communications of the MANET.



(a) The changes of event delay with the increase of consensus nodes

(b) The changes of data dropped rate with the increase of consensus nodes

Figure 10. The changes of event delay and data dropped rates with the increase of consensus nodes.

Therefore, our scheme using hashgraph and the blacklist is efficient in a MANET-based blockchain.

- Inter-sharding consensus

Scenario 1: Two shards are deployed, and each shard contains $four$ nodes. Node $node100$ moves from shard1 to shard2. The cross-shard transactions are generated by nodes in shards continuously.

Figure 11 shows the throughput of cross-shard transactions when *node*100 dynamically leaves shard1 and joins shard2. *Interval* indicates the generation interval of transactions, and *size* is the size of transactions. Nodes in shard2 can only receive the transactions between 1 m 32 s and 2 m 35 s, since *node*100 arrives at the overlap region of shard1 and shard2 at that time, relaying the transactions. Hence, dynamically joining and leaving shards contributes to inter-sharding consensus. When the transaction size is larger, its throughput drops, because *node*100 has limited resources and becomes the bottleneck. When the generation interval of transactions increases, the throughput increases as well. Hence, smaller sizes of cross-shard transactions, more frequency of nodes joining a shard and relatively fewer cross-shard transactions can improve the throughput of cross-shard transactions.

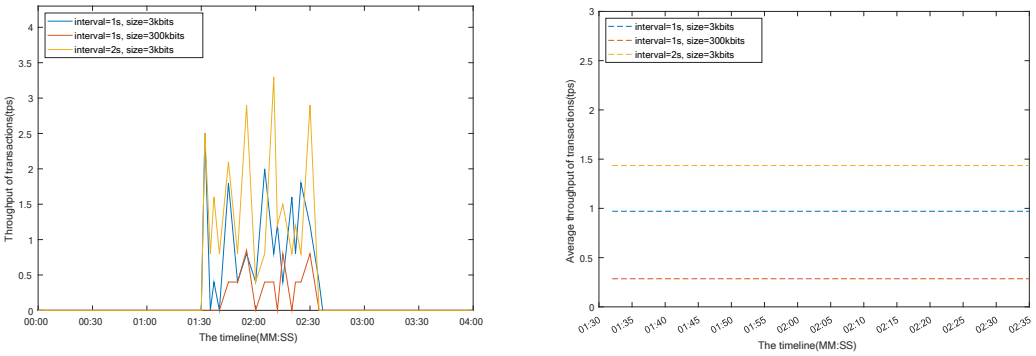


Figure 11. (**Left**) subfigure as the throughput of cross-shard transactions when a node dynamically leaves a shard and joins another shard. (**Right**) subfigure as the average of the throuput between 01:32 and 02:35 in the (**Left**) subfigure

Scenario 2: Six shards (shard1, shard2, shard3, shard4, shard5 and shard6) are deployed, and each shard contains 3 nodes. Node *node*100 moves from shard1 to shard2. In order to join shard2, *node*100 generates the request of a *cross-shard joined transaction* involving shard2 and shard1, the request involving shard2, shard1 and shard3, the request involving shard2, shard1, shard3 and shard4, the request involving shard2, shard1, shard3, shard4 and shard5, or the request involving shard2, shard1, shard3, shard4, shard5 and shard6.

Figure 12 shows the delay of the cross-shard transaction update in the number of different involved shards. Serial transaction validation means that before the cross-shard transaction updated, it is validated by involved shards one by one. Parallel transaction validation means that before the cross-shard transactionis updated, it is validated by related shards in parallel and then by shard2. The delay of valid cross-shard transactions in our scheme is better than that in serial transaction validation and parallel transaction validation, because our scheme is asynchronously tree-assisted and the delay is only related to shard2 *node*100 joining. Specifically, the update in our scheme is related to one consensus round, and only when the updated *cross-shard joined transaction* is used is it validated via the tree. Hence, we can see that the delay of cross-shard transactions in our scheme remains stable with the increase of involved shards. If the number of involved shards is six, the average delay of involved shards is 5387/6 = 897.8 ms, larger than the delay in our scheme of 855 ms, because the former takes the transmission delay among shards into account. Thus, our scheme is more suitable for opportunistic communications among shards.

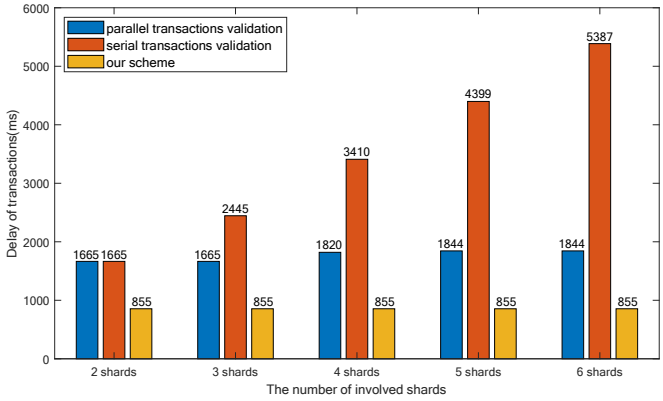


Figure 12. The delay of cross-shard transactions in the number of different involved shards.

8. Conclusions

In this paper, we have presented the behavior-based sharding hashgraph scheme for MANETs. Dishonest behaviors of nodes are recorded in the decentralized blacklist. With the assistance of the blacklist, the reputation values of candidate receivers for gossip information, candidate senders for gossip information, the requester for joining a shard, and a shard are evaluated. In this way, gossip information is sent to a reliable receiver, and gossip information from another reliable sender is received. It improves the efficiency of the hashgraph mechanism. In addition, the tree-assisted inter-sharding consensus is proposed to prevent Sybil attacks. Only the nodes with enough reputation values are allowed to participate in activities in a shard and without waiting. It also improves the efficiency of the node joining a shard. Moreover, the combination of shard recovery and reconfiguration based on shard reputation is proposed to prevent targeted attacks. The reputation values of a shard not only resist DDoS (Distributed Denial of Service) attacks of shard recovery but also improve the efficiency of shard reconfiguration through a dynamic epoch. Finally, we have shown that our proposed scheme can achieve some security properties, i.e., accountability, adaptability and decentralization. A lot of experiments are conducted to simulate the communication of consensus and disclose the effects of mobility on consensus. The results demonstrate that the proposed scheme is efficient.

In our future work, some real Epuck2.0 robots and an electronic sand table are used to implement a MANET-based blockchain in practical scenarios. Due to scarce wireless resources, robots need to compete for transmission channels. The distance between the communication parties and the interference from nearby robots also weaken the signal, increasing the latency of events to become famous witnesses. Hence, the hashgraph mechanism and gossip about gossip need to be improved further.

Author Contributions: Methodology, Y.H.; Software, Z.H.; Writing—original draft, R.L.; Writing—review & editing, G.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work is funded by the National Key-Area Research and Development Program of China (2018YFB1404402), Key-Area Research and Development Program of Guangdong Province(No. 2019B010137003), Guangdong Science & Technology Fund (No. 2016B030305006, No. 2018A07071702, No. 201804010314), and Guangzhou Science & Technology Fund (No. 201804010314), VeChain Foundation (No. SCNU2018-01).

Data Availability Statement: The research data are uploaded as an attachment along with the article.

Acknowledgments: I would like to give my heartfelt thanks to all the people who have ever helped me in this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Salman, T.; Zolanvari, M.; Erbad, A.; Jain, R.; Samaka, M. Security services using blockchains: A state of the art survey. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 858–880.
2. Kang, J.; Yu, R.; Huang, X.; Wu, M.; Maharjan, S.; Xie, S.; Zhang, Y. Blockchain for secure and efficient data sharing in vehicular edge computing and networks. *IEEE Internet Things J.* **2018**, *6*, 4660–4670.
3. Sharma, V.; You, I.; Jayakody, D.N.K.; Reina, D.G.; Choo, K.K.R. Neural-blockchain-based ultrareliable caching for edge-enabled UAV networks. *IEEE Trans. Ind. Inform.* **2019**, *15*, 5723–5736.
4. Feng, W.; Yan, Z. MCS-Chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain. *Future Gener. Comput. Syst.* **2019**, *95*, 649–666.
5. Bruzgiene, R.; Narbutaite, L.; Adomkus, T. MANET network in internet of things system. *Ad Hoc Netw.* **2017**, *66*, 89–114.
6. Morales, D.C.; Velloso, P.; Guerre, A.; Nguyen, T.M.T.; Pujolle, G.; Alagha, K.; Dua, G. Blockgraph proof-of-concept. In Proceedings of the SIGCOMM'21 Poster and Demo Sessions, Virtual Event, 23–27 August 2021; pp. 82–84.
7. Zhou, S.; Zhang, G.; Meng, X. LocTrust: A local and global consensus-combined trust model in MANETs. *Peer-to-Peer Netw. Appl.* **2022**, *15*, 355–368. [CrossRef]
8. Lwin, M.T.; Yim, J.; Ko, Y.B. Blockchain-based lightweight trust management in mobile ad-hoc networks. *Sensors* **2020**, *20*, 698. [PubMed]
9. Ilbeigi, M.; Morteza, A.; Ehsani, R. An infrastructure-less emergency communication system: A blockchain-based framework. *J. Comput. Civ. Eng.* **2022**, *36*, 04021041.
10. Liu, G.; Dong, H.; Yan, Z.; Zhou, X.; Shimizu, S. B4SDC: A blockchain system for security data collection in MANETs. *IEEE Trans. Big Data* **2020**, *8*, 739–752.
11. Rasool, S.; Iqbal, M.; Dagiuklas, T.; Ul-Qayyum, Z.; Li, S. Reliable data analysis through blockchain based crowdsourcing in mobile ad-hoc cloud. *Mob. Netw. Appl.* **2020**, *25*, 153–163.
12. Jiao, Z.; Zhang, B.; Zhang, L.; Liu, M.; Gong, W.; Li, C. A blockchain-based computing architecture for mobile ad hoc cloud. *IEEE Netw.* **2020**, *34*, 140–149.
13. Luo, J.; Shu, X.; Zhai, Y.; Fu, X.; Ding, B.; Xu, J. A Fast and Robust Solution for Common Knowledge Formation in Decentralized Swarm Robots. *J. Intell. Robot. Syst.* **2022**, *106*, 68.
14. Chatzopoulos, D.; Gujar, S.; Faltings, B.; Hui, P. Localcoin: An ad-hoc payment scheme for areas with high connectivity: Poster. In Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Paderborn, Germany, 5–8 July 2016; pp. 365–366.
15. Long, T.; Qu, S.; Li, Q.; Kang, H.; Fu, L.; Wang, X.; Zhou, C. Efficient block propagation in wireless blockchain networks and its application in Bitcoin. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 3349–3368.
16. Baird, L.; Harmon, M.; Madsen, P. Hedera: A public hashgraph network & governing council. *White Pap.* **2019**, *1*, 9–10.
17. Castro, M.; Liskov, B. Practical byzantine fault tolerance. In Proceedings of the OsDI, New Orleans, LA, USA, 22–25 February 1999; Volume 99, pp. 173–186.
18. Ongaro, D.; Ousterhout, J. The raft consensus algorithm. *Lect. Notes CS* **2015**, *190*, 2022.
19. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. SSRN. 2008. Available online: https://ssrn.com/abstract=3440802 (accessed on 10 January 2023).
20. Li, L.; Huang, D.; Zhang, C. An Efficient DAG Blockchain Architecture for IoT. *IEEE Internet Things J.* **2022**, *10*, 1286–1296.
21. Hong, Z.; Guo, S.; Li, P. Scaling Blockchain via Layered Sharding. *IEEE J. Sel. Areas Commun.* **2022**, *40*, 3575–3588.
22. Huang, H.; Peng, X.; Zhan, J.; Zhang, S.; Lin, Y.; Zheng, Z.; Guo, S. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding. In Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications, Online, 2–5 May 2022; pp. 1968–1977.
23. Zhang, X.; Xia, W.; Cui, Q.; Tao, X.; Liu, R.P. Efficient and Trusted Data Sharing in a Sharding-enabled Vehicular Blockchain. *IEEE Netw.* 2022, *early access*
24. Wang, Y.; Yuan, L.; Jiao, W.; Qiang, Y.; Zhao, J.; Yang, Q.; Li, K. A Fast and Secured Vehicle-to-Vehicle Energy Trading Based on Blockchain Consensus in the Internet of Electric Vehicles. *IEEE Trans. Veh. Technol.* **2023**, *72*, 7827–7843.
25. Benchi, A.; Launay, P.; Guidec, F. Solving consensus in opportunistic networks. In Proceedings of the 16th International Conference on Distributed Computing and Networking, Goa, India, 4–7 January 2015; pp. 1–10.
26. Luu, L.; Narayanan, V.; Zheng, C.; Baweja, K.; Gilbert, S.; Saxena, P. A secure sharding protocol for open blockchains. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, Vienna, Austria, 24–28 October 2016; pp. 17–30.
27. Wang, J.; Wang, H. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. In Proceedings of the NSDI, Boston, MA, USA, 26–28 February 2019; Volume 2019, pp. 95–112.
28. Amiri, M.J.; Agrawal, D.; El Abbadi, A. Sharper: Sharding permissioned blockchains over network clusters. In Proceedings of the 2021 International Conference on Management of Data, Xi'an, China, 20–25 June 2021; pp. 76–88.
29. Huang, C.; Wang, Z.; Chen, H.; Hu, Q.; Zhang, Q.; Wang, W.; Guan, X. Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding. *IEEE Internet Things J.* **2020**, *8*, 4291–4304.
30. Wang, L.e.; Bai, Y.; Jiang, Q.; Leung, V.C.; Cai, W.; Li, X. Beh-Raft-Chain: A behavior-based fast blockchain protocol for complex networks. *IEEE Trans. Netw. Sci. Eng.* **2020**, *8*, 1154–1166.

31. Yun, J.; Goh, Y.; Chung, J.M. DQN-based optimization framework for secure sharded blockchain systems. *IEEE Internet Things J.* **2020**, *8*, 708–722.
32. Zhang, M.; Li, J.; Chen, Z.; Chen, H.; Deng, X. An efficient and robust committee structure for sharding blockchain. *IEEE Trans. Cloud Comput.* 2022, *early access*.
33. Gao, N.; Huo, R.; Wang, S.; Huang, T.; Liu, Y. Sharding-Hashgraph: A High-Performance Blockchain-Based Framework for Industrial Internet of Things with Hashgraph Mechanism. *IEEE Internet Things J.* **2021**, *9*, 17070–17079.
34. Asheralieva, A.; Niyato, D. Reputation-based coalition formation for secure self-organized and scalable sharding in iot blockchains with mobile-edge computing. *IEEE Internet Things J.* **2020**, *7*, 11830–11850.
35. Zhang, X.; Xia, W.; Wang, X.; Liu, J.; Cui, Q.; Tao, X.; Liu, R.P. The block propagation in blockchain-based vehicular networks. *IEEE Internet Things J.* **2021**, *9*, 8001–8011.
36. Morales, D.C.; Velloso, P.B.; Laubé, A.; Nguyen, T.M.T.; Pujolle, G. A performance evaluation of C4M consensus algorithm. *Ann. Telecommun.* **2023**, *78*, 169–182.
37. Fu, X.; Wang, H.; Shi, P.; Zhang, X. Teegraph: A Blockchain consensus algorithm based on TEE and DAG for data sharing in IoT. *J. Syst. Archit.* **2022**, *122*, 102344.
38. OPNET Network Simulator. Available online: http://opnetprojects.com/opnet-network-simulator/ (accessed on 10 January 2023).